Ameba: A new topology optimization tool for architectural design

Article · November 2020

CITATIONS
3

READS
869

1 author:

Yi Min Xie
RMIT University
442 PUBLICATIONS **16,855** CITATIONS

SEE PROFILE

IASS 2018 BOSTON, JULY 16-20
CREATIVITY IN STRUCTURAL DESIGN

Proceedings of the IASS Symposium 2018
Creativity in Structural Design
July 16-20, 2018, MIT, Boston, USA
Caitlin Mueller, Sigrid Adriaenssens (eds.)

Ameba: A new topology optimization tool for architectural design

Qiang Zhou [a,*], Wei Shen [a], Jin Wang [a], Yi Yi Zhou [a], Yi Min Xie [a,b,*]

[a] XIE Archi-Structure Design (Shanghai) Co., Ltd., Shanghai 200092, China
august.zhou@xieym.com
[b] Centre for Innovative Structures and Materials, School of Engineering, RMIT University, Melbourne 3001, Australia
mike.xie@rmit.edu.au

Abstract

Structural form-finding using computational simulations is helpful for architects during the conceptual design stage. However, due to the complexity and slow-speed, conventional form-finding tools are difficult to use for architectural design. This paper presents a new Rhinoceros plug-in named Ameba, which is a topology optimization tool based on the BESO method and FEniCS open-source computing platform. Firstly, 2D or 3D elasticity problem variational formulation is given in this paper according to the FEniCS format. The code for solving the variational formulation is provided in the context of automated modelling using FEniCS. The computational framework of Ameba is then described and its general operational process is shown. The capability of Ameba is demonstrated by solving a variety of architectural form-finding examples. The computational time of models with different element numbers is compared. The examples show that Ameba is highly efficient and easy-to-use. Finally, two future extensions of the Ameba framework that are particularly useful for architectural form-finding are outlined.

Keywords: Ameba, Topology optimization, Form-finding, BESO, FEniCS

1. Introduction

Structural form-finding based on topology optimization can change the topology of initial structure fundamentally and thus generate a totally new structural form. Topology optimization aims to distribute the material inside a prescribed design domain such that the obtained structure has optimal properties and satisfies prescribed constraints. Architects and structural engineers can obtain high performance structural forms according to the design requirements during the conceptual design stage by using topology optimization [1].

Bi-directional Evolutionary Structural Optimization (BESO), which is one of the most commonly used topology optimization methods, has been applied to a wide range of structural design problems including stiffness and frequency optimization, nonlinear materials and large deformation, energy absorption, multiple materials, multiple constraints, periodical structures, and so on [2]. In the construction study of the Sagrada Familia Church by Mark Burry's team and Yi Min (Mike) Xie's team, the results of BESO were strikingly similar to Gaudi's original designs. Arata Isozaki, a renowned Japanese architect, in collaboration with structural engineer Mutsuro Sasaki, created a free-form structure for the Qatar National Convention Center using BESO. In 2015, the Shanghai-Fab-Union Space pavilion designed by Philip Feng Yuan's team used the BESO method to shape the core space.

The main idea of BESO method is to gradually remove the most inefficient material and add material to the most needed locations. All modern methods for topology optimization used in commercial and academic software are based on finite element methods. However, the existing topology optimization is not user-friendly for architects, and the computational time for large-scale topology optimization

problems is high. In this paper, a new Rhinoceros plug-in named Ameba is presented, which is a topology optimization tool based on the BESO method and FEniCS open-source computing platform. Rhinoceros is a commercial 3D computer graphics and computer-aided design application software package, which is widely used by architects due to its ease of use and ability to create complex models and its parametric modelling tool called Grasshopper.

In the following sections the variational formulation of 2D or 3D elasticity problem is given according to the FEniCS format. The code for solving the variational formulation is provided in the context of automated modelling using FEniCS. The computational framework of Ameba is then described and its general operational process is shown. Finally, the performance of Ameba is demonstrated by solving a variety of architectural form-finding examples.

2. FEM implementation

2.1. Elasticity problem variational formulation

The equations govern small elastic deformations of a body Ω can be written as

$$-\nabla \cdot \sigma = f \text{ in } \Omega, \tag{1}$$

$$\sigma = \lambda tr(\varepsilon) I + 2\mu\varepsilon, \tag{2}$$

$$\varepsilon = \frac{1}{2}(\nabla u + (\nabla u)^{\mathsf{T}}). \tag{3}$$

where σ is the stress tensor, f is the body force per unit volume, λ and μ are Lame's elasticity parameters for the material in Ω, I is the identity tensor, tr is the trace operator on tensor, ε is the strain tensor, and u is the displacement vector field. We have here assumed isotropic elastic conditions.

After some derivation[4], we can summarize the variational formulation as: find $u \in V$ such that

$$a(u,v) = L(v) \quad \forall v \in \hat{V}, \tag{4}$$

where

$$a(u,v) = \int_\Omega \sigma(u) : \varepsilon(v) \mathrm{d}x, \tag{5}$$

$$\sigma(u) = \lambda(\nabla \cdot u) I + \mu(\nabla u + (\nabla u)^{\mathsf{T}}), \tag{6}$$

$$\varepsilon(v) = \frac{1}{2}(\nabla v + (\nabla v)^{\mathsf{T}}), \tag{7}$$

$$L(v) = \int_\Omega f \cdot v \mathrm{d}x + \int_{\partial\Omega_T} T \cdot v \mathrm{d}s. \tag{8}$$

v is a vector test function, and $\hat{V}$ is a vector-valued test function space. When the boundary consists of multiple parts, (8) can be expressed as:

$$L(v) = \int_\Omega f \cdot v dx + \sum_i \int_{\Gamma^i} T_i \cdot v ds \tag{9}$$

2.2. FEniCS implementation

In the following steps, several functions will be used, such as *Mesh, VectorFunctionSpace, TrialFunction, TestFunction, Function* and so on. These functions should be imported from the FEniCS library at the beginning of the codes in Python. The *fenics* module contains all functions. We can start with:

*from fenics import **

The finite elements mesh can be created from a geometry or a given filename in FEniCS. In order to interface the mesh generated by other software, creating mesh from a filename is available. The file should contain mesh data stored in DOLFIN XML format:

mesh = Mesh(mesh_file_name)

After the mesh has been created, a finite element function space *V* can be defined:

V = VectorFunctionSpace(mesh, "CG", 1)

This means that the element is the standard P1 linear Lagrange element, which is a triangle with nodes at the three vertices in 2D or a tetrahedron with nodes at the four vertices in 3D.

In mathematics, the trial spaces V and the test spaces $\hat{V}$ are distinguished. The only difference in the present problem is the boundary conditions. In FEniCS the boundary conditions are not specified as part of the function space, so it is sufficient to work with one common space *V* for both the trial and test functions.

u=TrialFunction(V)

v=TestFunction(V)

The next step is to specify the boundary condition: $u = u_D \; on \; \partial\Omega$. This is done by

bc = DirichletBC(V, u_D, expression, method= "pointwise")

where *V* is the function space, and *u_D* is an expression defining the solution values on the boundary. The typical construction is

u_D = Expression(formula, degree=1)

The *formula* must be written by C++ syntax. The second argument *degree* is a parameter that specifies how the expression should be treated in computations. On each local element, FEniCS will interpolate the expression into a finite element space of the specified degree.

The third argument *expression* to *DirichletBC* is a C++ string defining which points belong to the part of the boundary.

Often it is more practical to use multiple boundary conditions, one for each subdomain of the boundary. These boundary conditions such as *bc1, bc2…* can be collected in a list which will be passed to the solve function to compute the solution:

bcs = [bc1, bc2]

Before defining the variational problem we have to specify the load term f and T. Because the term f is usually over the whole domain, *f* can be represented as a *Constant*:

f = Constant((fx, fy)) or f = Constant((fx, fy, fz))

While the term *T* is usually over multiple domains of boundary and its value is different in each domain, *T* can be represented as different *Constant*:

T_i *= Constant((*Tx_i, Ty_i*)) or* T_i *= Constant((*Tx_i, Ty_i, Tz_i*))*

Before defining the bilinear and linear forms $a(u, v)$ and $L(v)$, the strain tensor $\varepsilon(u)$ and stress tensor $\sigma(u)$ must be expressed:

def *epsilon(u):*

    return *the stain expression of UFL language*

def *sigma(u):*

    return *the stress expression of UFL language*

We now have all the ingredients we need to define the variational problem:

*a = inner(sigma(u), epsilon(v))*dx*

*L = dot(f, v)*dx + dot(T1, v)*ds(1) + dot(T2, v)*ds(2)+…++dot(Tn, v)*ds(n)*

Note that these two line Python codes are very similar with the variational formulation, which is the key strength of FEniCS.

Having defined the finite element variational problem and boundary condition, we can now ask FEniCS to compute the solution:

u = Function(V)

solve(a == L, u, bcs)

As soon as the displacement u is computed, various field outputs or the functions of *u* can be computed. For example, we will compute the von Mises stress defined $\sigma_M = \sqrt{\frac{3}{2}s:s}$ where *s* is the deviatoric tensor

$$s = \sigma - \frac{1}{3}tr(\sigma)I$$

There is a one-to-one mapping between these formulas and the FEniCS code:

*s = sigma(u) - (1./3)*tr(sigma(u))*Identity(d)*

*von_Mises = sqrt(3./2*inner(s, s))*

The von_Mises variable is now an expression that must be projected to a finite element space:

V = FunctionSpace(mesh, 'CG', 1)

VON_MISES = project(von_Mises, V)

The strain energy density $\nu_\varepsilon = \frac{1}{2}\sigma\varepsilon$ also can be computed:

*strain_energy_density = 0.5*inner(sigma(u), epsilon(u))*

SED= project(strain_energy_density, V)

The integrand of the strain energy function $E = \frac{1}{2}\int_\Omega \sigma\varepsilon \mathrm{d}x$ is described in the UFL language in the same manner as we describe weak forms:

*strain_energy = 0.5*inner(sigma(u),epsilon(u))*dx*

SE= assemble(strain_energy)

Because the strain energy is a scalar value, the functional *strain_energy* is evaluated by calling the *assemble* function. FEniCS will recognize that the form has "rank 0" since it contains no trial and test functions and return the result as a scalar value.

2.3. Defining FEM function

The topology optimization should run the finite element analysis in each iteration step and the output field variables should be computed in every iteration step. Therefore, it is convenient if the finite element analysis processes are packaged as one class in Python. In addition, we can extend the class to implement other problems using FEniCS library, such as eigenvalue problem, heat Conduction, fluid mechanics, etc.

3. Ameba plug-in

3.1 Ameba framework

The main motivation for the development of the proposed topology optimization framework is to provide a highly efficient and easy-to-use tool for designers. To provide an overview of the framework, a diagram of the code components can be seen in Fig. 1. There are five layers in the diagram. The foundation of the framework is the FEniCS library. Each of the dashed boxes is a Python class. The `FEM` class, which is based on the FEniCS library, solves the 2D or 3D elasticity problem. It also contains some field variables output, such as von Mises stress, strain energy density, strain energy and displacement. The code segments of the `FEM` class have been introduced in detail in above article.

`Optimization` is a topology optimization class which is based on BESO method. The `Optimization` class also contains element filtering scheme which can overcome some numerical problems such as checkerboard and mesh-dependency in topology optimization. The exact implementation of BESO algorithm is out of the current paper, but can be found in references [2, 3]. The `ModelDatabase` class is used to read mesh data file that is generated by Rhinoceros software.

The `Input files` block contains mesh data file for the finite element analysis, topology optimization parameters and other configuration parameters. In fact, the current framework can work independently without preprocess platform. However, it may be difficult to edit the input files that have specified data format. To ensure that the proposed framework is friendly for designers, a preprocess platform must be introduced. Therefore, the Rhinoceros platform with Grasshopper, which is familiar to designers, is added in the top of the framework. In Rhinoceros, designers could build various 2D or 3D geometric models. Having established geometric model, designers can use Ameba plug-in in grasshopper to generate input files automatically. The meshing tool is Gmsh[5].

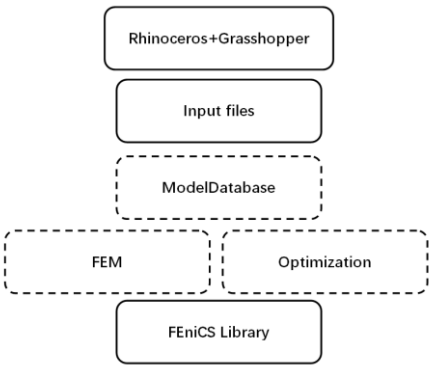


Figure 1: Layout of Ameba framework

3.2 BESO in Ameba

The soft-kill BESO method is used in Ameba and the parameters are $x_{\min}$=0.001, p=3. The filter scheme is used to suppress the checkerboard pattern. The element removal/addition process and the filter scheme are the same as the method described in Chapter 3 of [3]. To improve filtering efficiency, KD-tree algorithm is used to accelerate filtering.

The computation process of Ameba continues until the objective volume is reached and the following convergence criterion is satisfied.

$$error = \frac{|\sum_{i=1}^{N} C_{k-i+1} - \sum_{i=1}^{N} C_{k-N-i+1}|}{\sum_{i=1}^{N} C_{k-i+1}} \leq \tau$$

where k is the current iteration number, τ is a allowable convergence tolerance and N is an interger number. The default parameters are N =5, τ =1e-4.

3.3 General operation process

The operational process of Ameba plug-in is very similar to the use of finite element software. The simple steps are given below:

• step1: Create geometry model. The geometry model of design domain should be created firstly and then use `Mesh` component to generate mesh. The files' directory and the current project name should be defined.

• step2: Add supports. Users can select points, lines or curves on the geometrical boundary to define supports. Ux, Uy and Uz (for 3D) can be fixed or specified displacement value or released respectively.

• setp3: Add loads. The domains, which can be applied loads, are the same as supports' domains. Fx, Fy and Fz (for 3D) can be specified load value respectively.

• step4: Set optimization parameters. There are four parameters in this procedure, sensitivity number, constraint volume fraction, evolution ratio and filtering radius.

• step5: Add material. The FEM module supports linear elasticity solver at present. The linear elasticity materials can be applied on design domain.

• step6: Generate input file. When the 1-5 steps are completed and these components are linked to the `PreProcess` component, the input files will generate automatically. The input files will generate as soon as the component parameters change.

• step7: Solve. The cloud computing is used in Ameba plug-in. All the computing process is running in the cloud. And the result file of each completed iteration will send back to local specified folder during computing.

• step8: Display results. We can use `Display` component to display every iteration step topology optimization result by reading received result file.

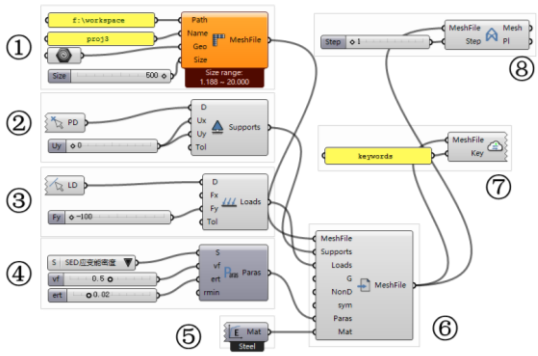


Figure 2: General operation process

The plug-in interface is shown in Fig. 3. Currently, Ameba is only suitable for minimum compliance problem and supports topology optimization of 2D and 3D arbitrary shape geometry. And symmetry constraint can be applied, i.e. symmetry line in 2D and symmetry plane in 3D.

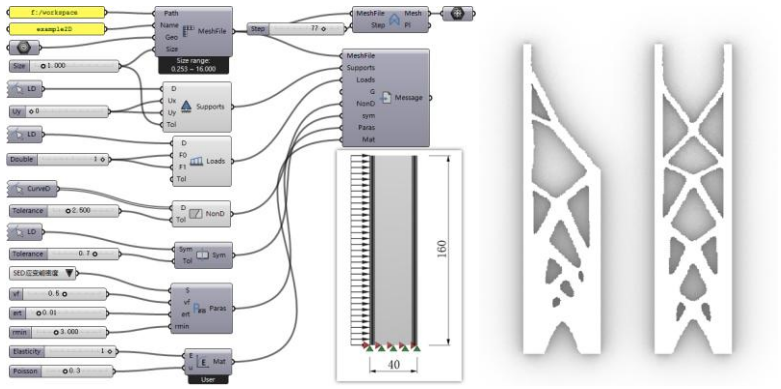
Figure 3: Ameba plug-in interface

4. Example and discussion

This section focuses on a series of example in 2D and 3D, illustrating how the Ameba plug-in is used in the architectural concept design. The Young's Modulus E=1 MPa and the Poisson's ratio v= 0.3 are assumed in each example. The cloud computing service has 4 CPU cores and 8GB memories.

4.1 High-rise building design

In high-rise building design, the manner in which material is distrubuted is usually significant for engineers to develop a lateral bracing system or create a conceptual design for structure members. An example of a typical high-rise building topology optimization is a cantilever beam problem which can be seen in Fig.4a. The geometric domain 40mm×160mm is created in Rhinoceros, and both vertical sides are defined as non-design domain. The domain is discretized into 16996 three node plane stress elements. A simple uniform wind load w=1N/mm is subjected to the left side and the bottom side is fixed. The following parameters are used: ER=1%, V^*=50%, $Rmin$=3mm. The design problem runs for 90 iterations for the model without symmetry constraint and 79 iterations for the one with symmetry constraint. The average time for each iteration is about 1.77s. Fig.4b and Fig.4c show the final design results.



(a) (b) (c)

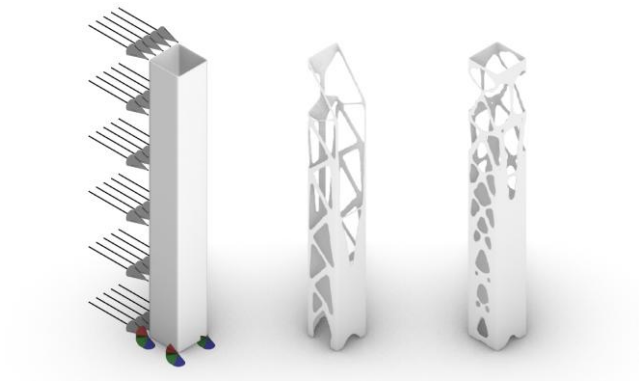
Figure 4: Topology optimization design for high-rise building:

(a) design domain subject to lateral load, (b) without symmetry constraint,

(c) with symmetry constraint

As we can see in Fig.4a, it is very simple to create a model for topology optimization using grasshopper in Rhino. The same modeling process can implement 3D topology optimization by replacing 2D components with corresponding Ameba 3D components.

Figure 5 illustrates the conceptual design of a 3D cantilever beam with box section. The design domain is 80×80×640 with a maximum size of 5 mm. The element type is four nodes tetrahedron element, and the total number is 49,344. A uniform wind load w=1N/mm^2 is subjected to the left surface and the bottom side is fixed. The following parameters are used: ER=1%, V^*=50%, Rmin=15mm. Fig.5b and Fig.5c give the resulting topologies at the last iteration. The design problem runs for 91 iterations for the model without symmetry constraint and 100 iterations for the one with symmetry constraint. The average time for each iteration is about 9.87s and 12.05s respectively.



(a) (b) (c)

Figure 5: Topology optimization design for 3D cantilever beam: (a) design domain subject to lateral load, (b) without symmetry constraint, (c) with symmetry constraint.

4.2 Spatial shell structure design

In spatial shell structure design, we can find the optimum design in the volume constraint which can be used as stiffener of the initial shell. Figure 6 shows the conceptual design of a triangle shell and a square shell. Both are fixed in foot and subjected to gravity load and lateral load. Some symmetry constraints are applied. The edge length of triangle shell and square shell is 100mm with the maximum size of 2mm. The total element number is 9,706 and 25,070 respectively. The following parameters are used: ER=2%, V^*=50%, $Rmin$=4mm. As the proportion of gravity load and lateral load changes, the optimization results are different. Fig.5 shows the resulting topologies of different load ratios. The

design problem runs for 44-93 iterations for the two models. The average time for each iteration is about 1.33s and 4.94s respectively.

(a) (b) (c) (d) (e) (f)

Figure 6: Topology optimization design for spatial shell structure: (a) design domain subject to gravity and lateral load, (b) 1.0 unit lateral load, (c) 1.0 unit gravity, (d) 1.0 unit gravity + 0.5 unit lateral load, (e) 1.0 unit gravity + 1.0 unit lateral load, (f) 1.0 unit gravity + 2.0 unit lateral load.

As we can see from above examples, a variety of conceptual designs could be generated within a few minutes. The process of creating a model for topology optimization is very simple by using Ameba plug-in. While topology optimization is useful for structural form-finding, usually the resulting topologies without any other constraints consist of complex geometries and poor materials layouts which are of limited value to real engineering projects due to construction difficulties. When the symmetry constraints are applied, the results are regular and present some periodicity which would reduce the complexity of the construction. In order to make the optimization results acceptable, there are much work to do in the future.

5. Conclusion

In this paper, a new topology optimization tool for architectural conceptual design is presented. The tool is based on grasshopper in Rhino and FEniCS open-source computing platform. It is easy-to-use by architects or engineers for generating architectural conceptual models. The capability and efficiency of Ameba are demonstrated by solving several examples of architectural form-finding. The Ameba plug-in can be downloaded at http://ameba.xieym.com.

The present work uses the structural compliance as the objective function. However, for architectural form-finding, other objective functions including deflection, stability and natural frequency may need to be considered. Moreover, in order to make the resulting topologies more applicable to engineering, various constraints should be imposed. These new functions will be implemented in Ameba next.

References

[1] Yuan F., Chai H., Xie Y.M., Towards a structural performance-based architectural desgin in digital age. Architectual Journal, 2017; 11; 001-008.

[2] Huang X., Xie Y.M., *Evolutionary Topology Optimization of Continuum Structures: Methods and Applications*, Chichester, Wiley, 2010.

[3] Zuo Z.H., Xie Y.M., A simple and compact Python code for complex 3D topology optimization. *Advances in Enigineering Software*, 2015; 85; 1-11.

[4] Logg A., Mardal K.A., Wells G.N. (editors), *Automated Solution of Differential Equations by the Finite Element Method*: *The FEniCS Book*, Heidelberg, Springer, 2011.

[5] Christophe G., Jean F.R., Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. International Journal for Numerical Methods in Engineering, 2009; 79(11); 1309-1331.